Workflow Adaptation as an Autonomic Computing Problem

Kevin Lee, Rizos Sakellariou, Norman W. Paton and Alvaro A. A. Fernandes
School of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, U.K
{klee, rizos, norm, alvaro}@cs.man.ac.uk

ABSTRACT

The performance of long running scientific workflows stands to benefit from adapting to changes in their environment. Autonomic Computing provides methodologies for managing run-time adaptations in managed systems. In this paper, we apply the monitoring, analysis, planning and execution (MAPE) model from autonomic computing to support the runtime modification of workflows with the aim of improving their performance. We systematically identify run-time adaptations and indicate how such behaviours can be captured using the MAPE model from the Autonomic Computing community. By characterising these as autonomic computing problems we make a proposal about how workflow adaptation can be achieved.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Graphs and networks; F.2.2 [**Nonnumerical Algorithms and Problems**]: Scheduling

General Terms

Management, Performance, Design, Experimentation

Keywords

Workflows, Autonomic Computing, Adaptation, Scheduling

1. INTRODUCTION

The autonomic computing community focuses on providing techniques for enabling autonomic behaviour for systems. The vision of the autonomic computing community [13] is to free up system administrators from the tasks of low-level system administration and optimisation. This applies equally to high-level management of system-wide policies and to the execution parameters of single programs on execution nodes. One of the main contributions of the autonomic computing community is a model that identifies how to capture autonomic behaviour for a given managed resource. This takes the form of a functional decomposition, known as MAPE, which consists of four stages in a pipeline, namely Monitoring, Analysis, Planning and Execution [12]. This provides a structure and methodology for developing adaptive systems.

There are many situations in workflow execution where adaptation could be of benefit to performance. Scientific workflows in particular introduce a challenge in execution due to their long-running and distributed nature. Here, adapting to environmental conditions can be of huge benefit, decreasing execution times by reacting to problems and opportunities positively. Besides adaptation to environmental conditions, there are other flavours of evolution for workflows. For example, business workflows represent business processes that evolve as the business changes [5]. However, such changes tend to be in response to external forces or changes in the policies of the business, rather than changes in the execution environment and as such they are out of scope in this paper.

To date, research aiming to improve workflow performance based on adaptive execution has relied mostly on bespoke techniques that are tied to particular workflow environments [9, 11], or appeals to a generic architectural model without characterising specific adaptations or the contexts in which they are likely to be effective [4]. This is detrimental to the creation and study of a corpus of environment-independent adaptive techniques that are suitable for workflow execution. Furthermore, casting workflow adaptation as an autonomic computing problem, allows us to take advantage of experiences in formulating adaptation in a systematic way.

In this paper, we use the MAPE framework to characterise performance-related adaptations in scientific workflows. We identify a range of possible adaptations for scientific workflows and show how MAPE can be used to clarify important issues and identify recurring themes.

The remainder of this paper is structured as follows. In Section 2, we provide some background on autonomic computing and outline the MAPE model. Section 3 presents our understanding of workflows and provides a concrete view with which to discuss them. Section 4 then presents a characterisation of possible adaptations in workflows. In Section 5, we discuss how the MAPE functional decomposition can be used to support the decision making process for workflow adaptation. Finally, we present our conclusions in Section 6.

2. AUTONOMIC COMPUTING

Autonomic computing is concerned with automating system management and optimising tasks that have tradition-

WORKS'07, June 25, 2007, Monterey, California, USA.

ally been done by hand by human managers or else by complex bespoke applications. Autonomic computing attempts to deal with this problem by using structured models that support run-time adaptation at all levels of system management [3, 12, 14, 15].

Perhaps the most widely adopted architecture for autonomic computing views adaptation of a system as being manageable by the use of autonomic managers [12]. These are effectively control loops that monitor the execution characteristics of the system and decide on necessary adaptations. An autonomic manager can be functionally decomposed into multiple stages of a control loop called the MAPE model, as illustrated in Figure 1 (taken from [12]).

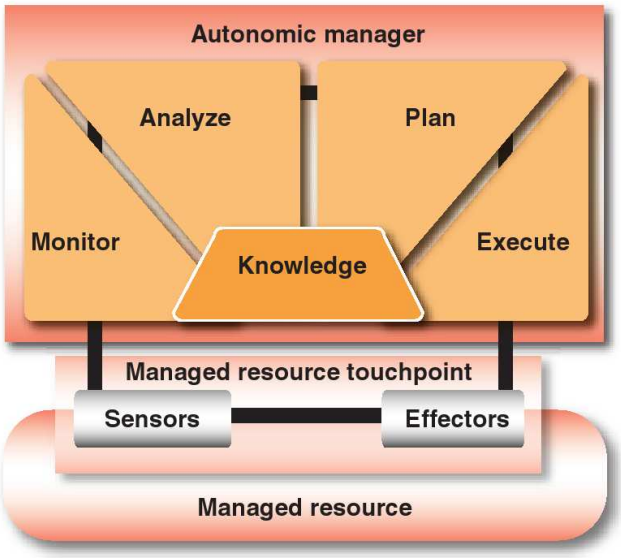

Figure 1: MAPE Autonomic Manager

The MAPE model is structured into two main software components, the *Managed Resource* and the *Autonomic Manager*. The *Managed Resource* is simply the system that is to be adapted, the assumption being that it is not initially adaptive and that it can be modified/extended to support run-time adaptation. The MAPE model requires two types of touch-points into and out of the *Managed Resource*, called *Sensors* and *Effectors*. *Sensors* allow the manager to collect data from the *Managed Resource*, and *Effectors* allow the manager to perform adaptations to the *Managed Resource*.

The second component of the MAPE model is the *Autonomic Manager*, which is partitioned into four distinct functions: *Monitor*, *Analyse*, *Plan* and *Execute*. These are the functions that make up the decision process needed to adapt a system. *Monitoring* uses the *Sensors* to produce monitoring events for consumption by *Analysis*. *Analysis* processes monitoring events to look for potential problems and opportunities. *Planning* decides, based on the system state and analysis messages, if it is appropriate to perform an adaptation. *Execution* utilises the *Effectors* to perform the adaptations requested by *Planning*. In addition, the MAPE model assumes some *Knowledge* of the system to be adapted.

Functionally decomposing the infrastructure required to manage adaptation allows the separation of concerns within the autonomic manager. This yields opportunities in the way adaptation infrastructures can be built. For example, in query processing (QP) in distributed settings [10], the separation of monitoring from the rest of the adaptation infrastructure allows multiple monitoring components to be instantiated.

3. WORKFLOWS

As a workflow that is ready to execute contains many concrete details, with each node being a concrete, executable addressing concrete files, this level of detail does not make it a helpful representation for use by the scientists that have to define their workflows. Therefore, workflows may be defined at different levels of abstraction. The well-known scientific workflow execution environments Pegasus [8], Triana [7] and Kepler [1] all describe workflows at a more abstract or logical level, which is then compiled to a lower-level workflow for execution.

In this paper, an abstract workflow describes the workflow at the level of tasks that perform transformations on data. By contrast, a concrete workflow describes the workflow at the level of actual services, file-based inputs and outputs. Abstract workflows have to be mapped to concrete workflows, which can then be scheduled onto the available resources and executed. This process is illustrated in Figure 2.

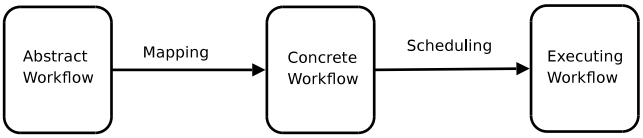

Figure 2: Workflow Compilation

An abstract workflow has nodes of the form $< n, i, o >$, where n is the id of a task, i is a set of input names and o is a set of output names. A task is a logical transformation of the input data to the output data; an input or output name denotes the data to be operated on. An edge in an abstract workflow is of the form $< t_1, o, t_2, i >$, and expresses a dependency between an output o of a task t_1 and an input i of a task t_2.

A concrete workflow has nodes of the form $< s, l, m, if, of >$, where s is the name of an service, l is the location of the service, m denotes whether or not it is a movable service, if is a set of input files and of is a set of output files. A service is available at location l; this can be either fixed (with respect to a resource that can run it) or movable. An input file is the location of the data to be fed to the service; an output file is the location where an output of the service is to be directed. An edge in a concrete workflow is of the form $< s_1, l_1, of_i, s_2, l_2, if_j >$ and expresses a dependency between an output of_i of a service s_1 at location l_1 and an input if_j of a service s_2 at location l_2.

Mapping an abstract workflow to a concrete workflow involves choosing appropriate services for tasks and finding data sources and output files. A task t is mapped to one or more concrete workflow nodes $n_1, \ldots, n_k$ depending on whether or not it is more efficient to split it across multiple nodes. At this stage, it may be appropriate to insert non-computational concrete nodes, such as those that perform stage-in and stage-out of data [8].

Scheduling a concrete workflow involves assigning each of

the services in the concrete workflow to execution nodes. Some services are fixed to their location and just need to be scheduled with respect to execution time at that location; other movable services may be moved to any available execution node and scheduled. A schedule is created of the form time t, service s, if files, of files, and execution node m for each concrete node. The schedule is used to execute each task on the resources at the appropriate time.

4. EXAMPLE WORKFLOW ADAPTATIONS

Workflow adaptations typically react to changes in the execution environment. Such changes may have a directly identifiable cause (for example, the addition or removal of nodes) or may be the result of more dynamic processes which affect the expected behaviour of the system (for example, an increase in the load of a resource that may cause a task to take longer than expected to complete). In addition, adaptations can be performed for different reasons, including *prospective* (to improve future performance), *reactive* (to react to previous results), and *altruistic* (to aid other areas of the system). Adaptations can also affect the workflow at different levels of granularity. They may apply to a single node, some nodes, or all of the workflow.

In general, adaptations can usefully be thought of as a revision of decisions made previously. Thus, based on the previous section, workflow adaptations can be classified as either *mapping* or *scheduling* adaptations.

4.1 Mapping Adaptations

Mapping adaptations are adaptations where the mapping from the abstract workflow to the concrete workflow changes depending on the environment. Based on the definitions in Section 3, possible mapping adaptations are to:

- **Change abstract node to concrete node mapping.** This changes the current abstract/concrete mapping to use different targets for s and l in the concrete node. In other words, this replaces the service(s) used to implement an abstract task with a different service s' or one from a different location l'.

- **Reduce the number of concrete nodes for an abstract task (task-reducing).** This is where a task t is mapped to fewer services $s'_1, \ldots, s'_n$ than before, with a view to, e.g., reducing network traffic and lag. This may be appropriate when, e.g., overheads per service instance are high and the execution resources in certain nodes have increased to the point that the task need no longer be split across as many service instances.

- **Increase the number of concrete nodes for an abstract task (task-splitting).** In this case, the number of services $s'_1, \ldots, s'_n$ onto which a task t is mapped is increased. This may be useful, e.g., if more resources are now available and overheads per service instance are low.

- **Remove an abstract node** A node $< n, i, o >$ can be removed if its removal does not compromise the correctness of the workflow. Removing an abstract node is possible, e.g., if its output o already exists, negating the need to perform the required processing to generate it.

- **Change data source/sink for a service.** This involves changing the concrete if and of files used as targets for mapping the abstract i and o. Changing if may be appropriate in the presence of replication if a change of data source/sink is judged to be beneficial given the current loads on resources.

4.2 Scheduling Adaptations

Adaptive scheduling involves the alteration of the scheduling policy in response to changes in the environment. Based on the definition in Section 3, possible scheduling adaptations are as follows:

- **Increase the level of parallelism of a service.** For a service s that is parallelisable, a potentially useful adaptation is to increase its parallelism level by scheduling it on more execution nodes at the same time. This may be possible depending on the availability of resources and the cost to increase the parallelism.

- **Decrease the level of parallelism of a service.** It may otherwise be appropriate to decrease the parallelism level of a service s by reducing the number of execution nodes the service is scheduled to execute on. This may be appropriate if, e.g., the resources for the service instance have to be reduced (e.g., because they must be reallocated to same other service with higher priority).

- **Restart service.** This may be appropriate if changes have been made to the configuration of a service s at a location l, and the service needs to be restarted to activate the changes (e.g., the amount of memory it can use is to be reduced).

- **Pause service.** This involves temporarily stopping the execution of a service s at a location l. The execution node can then be used to execute other service(s) with the released resources. Pausing services may be useful to control the overall performance of workflow execution in order to, e.g., enable other services to catch up.

- **Move service between execution nodes.** For services that can be executed on different nodes, it may be useful to move a service at a location l to location l'. This may be desirable if node resources fail, become scarce or become more expensive, or if it is beneficial in order to optimise the overall allocation of tasks onto resources. Such an adaptation will have a knock-on effect on concrete workflows with nodes that use s on l, requiring a mapping adaptation to *change abstract node to concrete node mapping*.

5. WORKFLOW ADAPTATIONS AS MAPE-BASED DECISION MAKING

The previous section illustrates the range of potential adaptations in workflow compilation and execution. This section shows how the autonomic computing approach briefly introduced in Section 2 provides a conceptual framework to structure and specify those adaptations, thereby providing help in dealing with the underlying performance management issues.

The *Autonomic Manager* in the MAPE model embodies a functional decomposition that is useful for structuring the decision making processes involved in supporting adaptations. The *Monitoring*, *Analysis*, *Planning* and *Execution* functional phases can be used to investigate the adaptive opportunities. Together, they provide a consistent, abstract viewpoint with which to express adaptation strategies for a given kind of *Managed Resource*. The remainder of this section takes each of the above functional phases in turn and shows how they can be used to characterise workflow adaptations.

5.1 Monitoring

In a scientific workflow of the type defined in Section 3, there is a set of generic data that can be monitored that will be of use in determining the state of the system, and some work has been undertaken to monitor workflow progress [17]. These monitoring data cab be characterised as relating to task state, execution environment state, service availability, etc.

The following is a list of task state data that it may be possible to monitor:

M.1 Progress of a service. Generally, this could rely on check points within the service, or a service may be able to provide an estimate of its progress (e.g., [6]).

M.2 Completion of a service. This could be a simple event that indicates that s has produced all of its of.

M.3 Data consumption rate of a service. This is a measure of the rate at which s is consuming data from if.

M.4 Data production rate of a service. This is a measure of the rate at which s is generating data for of.

A list of the useful data that it may be possible to monitor about the state of the environment is:

M.5 Available execution nodes. This could be a list of changes in the available execution nodes in the environment.

M.6 Load on an execution node. This is a measure of the load in a execution node. It could be one, or a tuple, or a composite of, e.g., the CPU load, the number of processes, and the free resources of the execution node.

M.7 Load on a network link. This is a measure of the usage of a network link, e.g., in terms of the available bandwidth.

M.8 Memory usage on an execution node. This is a measure of the usage of memory in a execution node.

Besides information about the state of the execution environment, data on the available services is also needed. The following is a list of useful data that it may be possible to monitor about service availability:

M.9 Available services. This could be a list of the services available as mapping targets for tasks in a workflow. The data could also include, e.g., the status of services currently deployed. Such data can be supplied by a service such as the Globus Replica Location Service [18].

M.10 Available data resources. This could be a list of the data resources available as mapping targets for inputs and outputs in a workflow. Note that if data resources expose service interfaces (as is the case with, e.g., [10]), service availability information may subsume data resource availability information.

Monitoring the available services is desirable to determine what services can best be used by the mapping stage of workflow execution. Likewise, by monitoring the available data sources the most appropriate source can be selected, e.g., the one with the fastest data rate.

Monitoring data is provided by the *Monitoring* component inside the *Autonomic Manager* by appeal to *Sensors* that need to be made available for instantiation in the *Managed Resource*. The *Monitoring* component has to decide the most appropriate interval for data retrieval in order that monitored data is recent enough to be useful while minimising performance overheads.

5.2 Analysis

It is the purpose of the *Analysis* component to use the available monitoring data to detect issues that may help decide whether some adaptation is desirable. Analyses are aimed at detecting either problems or opportunities (or both). The following is a list of potential problems that can be looked for using the monitoring data discussed in Section 5.1.

A.1 There is load imbalance. Load imbalance will occur if appropriate tasks are not scheduled to appropriate execution resources and the overall load fails to be sufficiently evenly distributed. This may be detectable by *Analysis* by using $M.1$, $M.2$, $M.3$ or $M.6$ data.

A.2 There is a bottleneck. This is where a node in the concrete workflow becomes a bottleneck that is potentially holding up its dependent nodes. Bottlenecks may be detectable by $M.1$, $M.3$ or $M.4$ data.

A.3 A workflow is likely to miss its expected completion time. To determine if an expected completion time is likely to be missed, the progress of each of the executing services can underpin an estimation of the overall completion time. For this, $M.1$ and $M.2$ data can be used and compared against the original schedule.

A.4 An execution node has failed. Detecting whether or not an execution node has failed may not be straightforward, but an indication may arise from temporal analyses over $M.5$-$M.8$ data. Alternatively, one could analyze $M.3$ and $M.4$ data to identify whether a service has consumed or produced any data over a given period of interest.

As well as detecting problems, *Analysis* also aims at detecting opportunities. A list of possible opportunities is as follows:

A.5 There is additional free capacity. This may be detectable by looking at changes in $M.5$-$M.8$ data.

A.6 There is a new service instance available. This may be done by analysis of $M.9$ data.

A.7 There is a new data resource available. This is the corresponding case, now on data resources and hence based upon $M.10$ data.

A.8 There are underutilised execution nodes. By looking at $M.1$, $M.3$, $M.4$, and $M.5$ data, it may be possible to detect that an execution nodes has spare capacity.

5.3 Planning

The purpose of *Planning* is to decide whether there are adaptations whose outcome is likely to be the beneficial removal of one or more problems (e.g., in $A.1$-$A.4$) or the beneficial capitalisation on an opportunity (e.g., in $A.5$-$A.8$). Based on the analyses in Section 5.2 and the adaptations identified in Section 4, a list of possible plans and the analyses they aim to address could be:

P.1 To make a workflow complete more quickly by increasing service parallelism. In response to problems detected by $A.2$ and $A.3$.

P.2 To make a workflow complete more quickly by rescheduling a service to a different execution node. In response to problems/opportunities detected by $A.1$, $A.2$, $A.3$, $A.4$ or $A.6$.

P.3 To make a workflow complete more quickly by replacing a service. In response to problems detected by $A.2$ and $A.3$.

P.4 To schedule a service to an alternative execution node. In response to problems detected by $A.4$.

P.5 To assign free execution nodes to a service. In response to problems/opportunities detected by $A.1$, $A.2$, $A.3$, $A.4$ or $A.6$.

P.6 To move services and take advantage of free network resources. In response to problems/opportunities detected by $A.1$, $A.2$, $A.3$, $A.4$ if supported by $M.7$ data.

P.7 To resort to faster data sources. In response to problems detected by $A.1$, $A.2$, $A.3$ if opportunity $A.7$ has also been detected.

Given the problems and opportunities detected by different analyses, the available capabilities reflected in the set of *Effectors* in *Managed Resource* and the contents of the *Knowledge* base in the *Autonomic Manager*, the *Planning* component has to decide what to do. It has to calculate the consequences of performing any adaptations, including the possible benefits, any negative effects and the cost of carrying out the candidate adaptation, for each adaptation it contemplates. In most cases, the outcome of *Planning* is an agenda of actions that the *Execution* component can carry out.

For example, in distributed QP (say, [10]), if a hash join is partitioned to run in parallel in several nodes and load imbalance is detected, then the *Planning* component would not only compute a new distribution policy, but it would also have to stipulate the sequence, source and target of movements of state (in this case, the partial contents of the hash tables being used) that may be required.

5.4 Execution

In the MAPE model, carrying out the plan is the responsibility of the *Execution* component. This may involve, e.g., the implementation of protocols for suspending workflow execution in preparation for potentially intrusive changes, invoking the available *Effectors* (whose existence and capabilities the actions available to the *Planning* component presume), and then resuming workflow execution.

5.5 Example Application

To illustrate how using the MAPE model to support adaptation can work in practice, we use an example from the DAG scheduling domain. Scheduling DAGs completely and statically fails to take into account any environmental changes during execution. Such changes may give rise to both problems and opportunities with respect to the statically computed schedule. Rescheduling while the DAG is executing provides an opportunity to contend with some of problems and to pounce on some of the opportunities. However, it is important to have control on the frequency of such attempts, as (re)scheduling is computationally expensive. A dynamic rescheduling approach is taken in [16]. Task slack is used to limit the frequency of reschedules. Task slack is the amount the execution of a task can slip without affecting the overall execution time for the DAG.

This approach can be seen in the context of this paper as a scheduling adaptation. We can cast the problem as an autonomic computing one by deriving the requirements at each stage of the MAPE model. Task slack is determined by calculating the amount of time a task can slip before the overall DAG schedule is delayed. The required monitoring information to calculate task slack is one or more of $M.1$, $M.2$, $M.3$ and $M.4$ as well as the original schedule.

The *Monitoring* component delivers this information as a data stream (alternatively, as a stream of measurement events). The *Analysis* components processes this data stream to detect the condition in which the delay in the execution of a task is greater than its slack (i.e., the delay in the execution of the task is likely to lead to a delay in the completion of the workflow). The requirement is therefore for an $A.3$-type of analysis. We have expressed this analysis using the CQL [2] query language, i.e., as a continuous query of a data stream. If a potentially disruptive delay is detected in the executing schedule S, the query emits a result characterising the problem. In this case, the *Analysis* component submits to the *Planning* component a request to attempt rescheduling (so, in this case, planning reduces to the run of an algorithm and a comparison, to check whether the new schedule S' eliminates (or alleviates) the problem detected). If it does, then the pair (S, S') allow the actions to be computed that effect the transition within the environment from S to S'. The *Execution* component deploys the changes, suspending, intruding and resuming as appropriate. The process of autonomically managing DAG execution can continue while beneficial outcomes can be expected from adapting.

5.6 Discussion

Section 5 shows how it is possible to use the MAPE model to structure the decision process for performing workflow adaptations. The *Monitoring*, *Analysis*, *Planning*, *Execution* decomposition focuses on the important aspects of adaptation and aids the decision making process. Furthermore, Section 5.5 discusses how MAPE can be used to support a

type of DAG rescheduling. In comparison to the approach discussed in [16] it also offers the potential for a decision-making process that is more capable of contending with the inherent complexities of workflow execution.

Using the MAPE model has a number of requirements for both the *Autonomic Manager* and the *Managed Resource*. All of the functions of MAPE discussed in this section require some domain knowledge of the *Managed Resource* to be available in the *Autonomic Manager*. For example, analysis detecting bottlenecks must know the concrete workflow dependencies of each node. In addition, the *Managed Resource* needs to provide the support of *Sensors* and *Effectors*.

However, this level of support would be necessary no matter the structure of the *Autonomic Manager*. The advantages of using the MAPE approach are that it provides a consistent and logical model to support adaptive systems and also provides a unifying framework to integrate systems with adaptive functionality.

6. CONCLUSIONS

In this paper we have argued that the autonomic computing approach to designing and specifying adaptive systems is suitable for adaptive workflow execution. We have suggested how workflow adaptations can be described in terms of the functional decomposition of autonomic managers into monitoring, analysis, planning and execution components. This provided a structure to discuss the decision-making process for deciding on and deploying adaptations, from the data needed, to the analyses required to detect problems and opportunities, to the construction and consideration of candidate plans, to the carrying out of the latter in the context of protocols that impose a discipline on potentially intrusive adaptations. Early indications suggest that workflow adaptation can be supported in a systematic and consistent way through the use of autonomic managers. In future work, we intend to further formalise our approach, design and implement software tools for developing MAPE-based adaptive systems, and evaluate the usefulness and performance of this framework using real-world scientific workflows.

7. REFERENCES

[1] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows, 2004.

[2] A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

[3] D. Balasubramaniam, R. Morrison, G. Kirby, K. Mickan, B. Warboys, I. Robertson, B. Snowdon, R. M. Greenwood, and W. Seet. A software architecture approach for structuring autonomic systems. In *DEAS '05: Proc 2005 Workshop on Design and Evolution of Autonomic Application Software*, pages 1–7. ACM Press, 2005.

[4] P. Buhler, J. M. Vidal, and H. Verhagen. Adaptive workflow = web services + agents. In *Proceedings of the International Conference on Web Services*, pages 131–137. CSREA Press, 2003.

[5] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 438–455, 1996.

[6] S. Chaudhuri, V. R. Narasayya, and R. Ramamurthy. Estimating progress of long running sql queries. In *SIGMOD Conference*, pages 803–814, 2004.

[7] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang. Programming scientific and distributed workflow with triana services. *Concurrency and Computation: Practice & Experience*, 18(10):1021–1037, 2006.

[8] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Metha, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39, 2003.

[9] Y. Gil, E. Deelman, J. Blythe, C. Kesselman, and H. Tangmunarunkit. Artificial intelligence and grids: Workflow planning and beyond. *IEEE Intelligent Systems*, 19(1):26–33, 2004.

[10] A. Gounaris, J. Smith, N. W. Paton, R. Sakellariou, A. A. A. Fernandes, and P. Watson. Adapting to changing resource performance in grid query processing. In *Data Management in Grids*, pages 30–44. Springer, 2005.

[11] T. Heinis, C. Pautasso, and G. Alonso. Design and evaluation of an autonomic workflow engine. In *Proc. ICAC*, pages 27–38. IEEE Press, 2005.

[12] B. Jacob, R. Lanyon-Hogg, D. K. Nadgir, and A. F. Yassin. *A Practical Guide to the IBM Autonomic Computing Toolkit*. IBM Redbooks, 2004.

[13] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[14] M. Litoiu, M. Woodside, and T. Zheng. Hierarchical model-based autonomic control of software systems. In *DEAS '05: Proc 2005 Workshop on Design and Evolution of Autonomic Application Software*, pages 27–33. ACM Press, 2005.

[15] S. M. Sadjadi, P. K. McKinley, and B. H. C. Cheng. Transparent shaping of existing software to support pervasive and autonomic computing. In *DEAS '05: Proceedings of the 2005 workshop on Design and evolution of autonomic application software*, pages 1–7, New York, NY, USA, 2005. ACM Press.

[16] R. Sakellariou and H. Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. *Scientific Programming*, 12(4):253–262, December 2004.

[17] H.-L. Truong, P. Brunner, T. Fahringer, F. Nerieri, R. Samborski, B. Balis, M. Bubak, and K. Rozkwitalski. K-wfgrid distributed monitoring and performance analysis services for workflows in the grid. In *Proc. 2nd Intl Conf on e-Science and Grid Computing*. IEEE Computer Society, 2006.

[18] S. Vazhkudai, S. Tuecke, and I. Foster. Replica selection in the globus data grid. In *CCGRID '01: Proc 1st International Symposium on Cluster Computing and the Grid*, page 106. IEEE Computer Society, 2001.